REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
07/05/2016

2. REPORT TYPE
Final Technical Report

3. DATES COVERED *(From - To)*
04/01/2012 - 03/31/2016

4. TITLE AND SUBTITLE
Early Student Support for Application of Advanced Multi-Core Processor Technologies to Oceanographic Research

5a. CONTRACT NUMBER
N00014-12-1-0298

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Alex Wiggins

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Oregon State University
308 Kerr Administration Building
Corvallis, OR 97331-8507

8. PERFORMING ORGANIZATION REPORT NUMBER
ONRBD025

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ONR Regional Office Seattle-N63374
300 Fifth Avenue, Suite 710
Seattle, WA 98104

10. SPONSOR/MONITOR'S ACRONYM(S)
~~ONE~~ ONR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
N63374

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited Distribution

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
UNCLASS	UNCLASS	UNCLASS		4	William Dillon
					19b. TELEPHONE NUMBER *(Include area code)* 541-737-5629

Optimizing Resource Use and Resilience in an Embedded Heterogeneous Cluster

We created an intelligent, reliable, and efficient compute scheduling and management system aboard an AV by employing a cluster of embedded computing systems to improve upon the following areas: Vehicle adaptability (i.e. re-planning and reconfiguration), computing, and technology adaptation. This work deviates from the current methods of using independent systems that are activated as needed for vehicle sensing and computing. Instead, a centralized system where resources are placed in a shared pool and tasks are dispatched and executed when availability and functionality permits, according to the task classifications and user described priorities.

This system provides the ability for rapid validation and upgradability of new hardware for vehicle computing systems. Using OpenCL task execution is not necessarily dependent on specific hardware, therefore computing resources are treated as a commodity and not a set of discrete devices. This change improves the ability of the cluster scheduler to handle performance measurements and task distribution. For each task a user provided description of the performance requirements, dependencies, and importance are used in conjunction with a self-classification to help the scheduler decide which tasks can and will be executed. These descriptions also help in determining which resources to use to maximize compute power and battery efficiency. This allows for domain scientists to spend their time and effort working on the science behind the sensing and measurements while the classifier and scheduler determines when and where the computation is performed.

This work also improves vehicle reliability by self-managing compute device failures and configuration based on resource availability. When a device is discovered that has failed or isn't performing as expected, it is removed from service and a new optimal computing setup is calculated and applied. Being able to recalculate and redistribute in-situ allows for improved vehicle deployment durations. As long as the resources are available to continue performing the tasks at hand, the new layout is used and the mission continues as expected without the need to any user intervention. This re-configurability also allows for re-planning on mission modifications and other vehicle changes that can impact a deployment. The following sections explore the current state-of-the-art research in the areas described above and how this work improves upon them in regards to embedded AV computing.

The intelligence and adaptability of this platform was realized by utilizing a Q-Learner (an AI based algorithm) coupled with a simulator for exploring device and kernel interactions. Using a "kernel classifier" that characterizes the performance and energy characteristics of an arbitrary algorithm a device vs kernel interaction description is automatically built. Coupling this characterization with a programming language (of sorts) that lets end-users define their tasks in a way that can generate the necessary structures, interrelationships, and ML constants automatically. Feeding this into the simulator and Q-Learning algorithm produces and state-action table that is used to enhance the reliability and performance of a computing cluster aboard an AV. Using this information, the scheduler can then compute a schedule in-situ and reconfigure

the compute system to match the expected layout using a custom set of hardware power sensing and relay boards.

Sensor Stream

We also created a near real-time data-store and data-distribution platform for devices streaming time-series data (i.e. temperature, audio, and video). We are called this system *SensorStream.* SensorStream was created to provide a pipeline for data collection, storage, processing, and distribution, allowing researchers to easily work with remote data acquisition devices. This platform is intended for sensor driven data collection and observation. This system can be utilized by devices ranging from simple environmental sensors measuring temperature and humidity, to complex Unmanned Aerial Vehicles (UAV) and Unmanned Underwater Vehicles (UUV) with side-scan sonar, high-definition video, and image sensors, or even phone applications that facilitate citizen science.

Currently, this workflow follows the typical pipeline; sense, analyze, model, and respond - all conventionally done by experts, offline, or using expensive proprietary software, and requiring large investments in hardware infrastructure and administration. This workflow becomes even more complicated with the number of devices and sensors researchers are using now. Current strategies for remote data collection call for new code solutions for each new sensor or platform. Each sensor and associated driver output their data in different formats using different specifications. This is not a scalable state of affairs for deploying large numbers of homogenous sensor platforms, let alone large numbers of heterogeneous platforms.

The SensorStream framework aims to simplify this process end to end, by presenting a unified cross-platform architecture, coupled with remotely accessible self-describing data. SensorStream accelerates the research process by reducing the latency in provisioning and deploying sensor packages and by moving from an offline only data collection and processing model to a near-real-time acquisition and analysis paradigm.

There are three key pieces to this project: the software for sensor systems to create data collection devices, the data store and distribution infrastructure, and the visualization and data exploration for the collected data. This follows the same path of sense, analyze, model, and respond, but attempts to do it in near real-time, eliminating the need for offline processing and expensive software packages that are used in today's pipelines.

For the device software, we developed two different types of packages. The first was a set of libraries in a variety of languages (Ruby, C#.NET, JavaScript, and C to date) to interface with our data storage and distribution application programming interfaces (APIs). The second is a complete software package for use on almost any system running Linux, from desktops systems to embedded devices (ARM, x64, and x86 processor architectures). This software handles everything from data collection, using an abstraction allowing for the creation of custom drivers that can utilize variety of communications protocols (i.e. UART, I2C, and SPI), through the

handing off of the data to the server APIs. By providing a common set of tools for constructing sensor platforms and rapidly collecting data we allow scientists to spend more time extracting meaning from the data that they collect. This could significantly improve the rate of expansion of scientific knowledge, and with less cost.

The server side APIs handle all the metadata, data storage, backups, and distribution. This architecture is the heart of our SensorStream system. The front-end APIs for this service run in Windows Azure on IIS 8. The IIS instance is responsible for interacting with the backend data store, caching, and processing as well as distributing data in clients in near real-time. The backend data store was implemented using Cassandra in order to enable this services to be scalable and redundant. Cassandra is a distributed database management system that allows for the service to scale out as needed, adding more servers to increase overall capacity and throughput in the cluster, and increasing the replication factor if needed to increase the throughput for "hot" data that is continually being accessed. In addition to scaling out with large distributed systems, Cassandra can also be used on smaller embedded systems and single nodes (at the cost of redundancy) allowing the service to deployed to micro-instances.

The final piece of this work is the visualization and analysis tools. These are implemented as a website in order to allow the largest number of people to access them from a wide variety of devices. This site includes the ability to retrieve data in tabular and graph form helping researches and users make sense of the collected data. This site also implements streaming data allowing users to create real-time dashboards and monitor their deployed sensor platforms. Personalized and shareable dashboards are the next step in this section, which will enable researchers to share their view of the data in real-time.